Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №3
на тему

АСИММЕТРИЧНАЯ КРИПТОГРАФИЯ. КРИПТОСИСТЕМА РАБИНА

Выполнил: студент гр.253502 Канавальчик А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Цель данной работы – изучить теоретические основы и разработать программную реализацию асимметричной криптографической системы Рабина, обеспечивающей конфиденциальность данных при передаче и хранении информации.

В ходе работы предстоит решить следующие задачи:

1 Изучить теоретические основы криптосистемы Рабина:

– Проанализировать математические основы системы: проблему извлечения квадратных корней по модулю составного числа, сложность факторизации больших чисел.

– Исследовать алгоритмы генерации ключевой пары: требования к простым числам p и q, вычисление n = p * q.

– Изучить процессы шифрования и дешифрования: вычисление квадрата по модулю, использование китайской теоремы об остатках для нахождения четырёх корней.

2 Разработать программную реализацию алгоритма на языке *Python*:

– Создать модуль генерации ключей, обеспечивающий формирование закрытого ключа (p, q) и открытого ключа (n) в соответствии с требованиями криптостойкости.

– Реализовать модуль шифрования данных, выполняющий преобразование сообщения m в шифртекст c = m^2 mod n.

– Реализовать модуль дешифрования, восстанавливающий исходное сообщение с применением закрытого ключа и механизма выбора истинного текста из четырёх вариантов.

3 Протестировать корректность работы реализации:

– Проверить корректность шифрования и дешифрования на различных тестовых наборах данных, включая крайние случаи.

– Оценить практическую применимость системы на примере обработки текстовых файлов.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Асимметричная криптография представляет собой метод шифрования, при котором для операций зашифрования и расшифрования используются разные ключи: открытый и закрытый. Данный подход обеспечивает безопасную передачу информации без необходимости предварительного обмена секретными ключами.

Криптосистема Рабина является одной из асимметричных криптосистем, безопасность которой основана на вычислительной сложности задачи извлечения квадратных корней по модулю составного числа. Эта проблема эквивалентна задаче факторизации больших целых чисел, что делает систему криптостойкой при условии использования достаточно больших простых чисел.

Основу криптосистемы Рабина составляет операция возведения в квадрат по модулю n = p * q, где p и q – большие простые числа, удовлетворяющие условиям $p \equiv 3 \pmod 4$ и $q \equiv 3 \pmod 4$. Открытым ключом является число n, а закрытым ключом – пара чисел (p, q).

Процесс шифрования заключается в вычислении c = m^2 mod n, где m – исходное сообщение. Для дешифрования требуется решить квадратное уравнение по модулю n с помощью китайской теоремы об остатках. В результате получаются четыре возможных корня, один из которых является истинным сообщением. Для однозначного восстановления текста применяются методы добавления избыточности или проверочных битов.

Криптосистема Рабина обладает высокой стойкостью к криптоатакам, так как взлом системы требует разложения модуля n на множители. По скорости шифрования она превосходит алгоритм *RSA* благодаря использованию единственной операции возведения в квадрат. Однако необходимость выбора истинного сообщения из четырех вариантов требует дополнительных механизмов верификации.

Система предназначена для применения в защищенных системах передачи данных, где требуется обеспечение конфиденциальности и аутентичности информации. Она может использоваться в государственных и коммерческих организациях для защиты критически важных данных.

3 ХОД РАБОТЫ

В ходе выполнения лабораторной работы было разработано программное средство для криптографической защиты текстовых данных на основе асимметричного алгоритма Рабина. Реализация поддерживает полный цикл операций: генерацию ключевой пары, шифрование и дешифрование информации.

Программный интерфейс реализован в интерактивном консольном формате с использованием объектно-ориентированного подхода. Основной функционал разделен на три класса:

1) *RabinCryptosystem* – реализует криптографические алгоритмы;
2) *FileHandler* – обеспечивает работу с файловой системой;
3) *RabinApp* – управляет пользовательским интерфейсом.

Программа поддерживает генерацию ключей длиной 512 бит и более, что соответствует современным требованиям криптостойкости. Для проверки простых чисел используется алгоритм из библиотеки *sympy*. Ключевая пара сохраняется в текстовом файле для последующего использования.

Процесс шифрования включает преобразование текстового сообщения в числовое представление с использованием кодировки *UTF-8* и выполнение операции возведения в квадрат по модулю N. Дешифрование реализует алгоритм нахождения четырех квадратных корней с применением китайской теоремы об остатках и расширенного алгоритма Евклида.

Для верификации результатов программа автоматически определяет корректный корень путем попытки декодирования всех четырех возможных вариантов. Это позволяет устранить основной недостаток криптосистемы Рабина - неоднозначность дешифрования.

Наглядное представление работы программы и результаты выполнения операций демонстрируются на рисунке 3.1, где отображен процесс взаимодействия с программным средством.

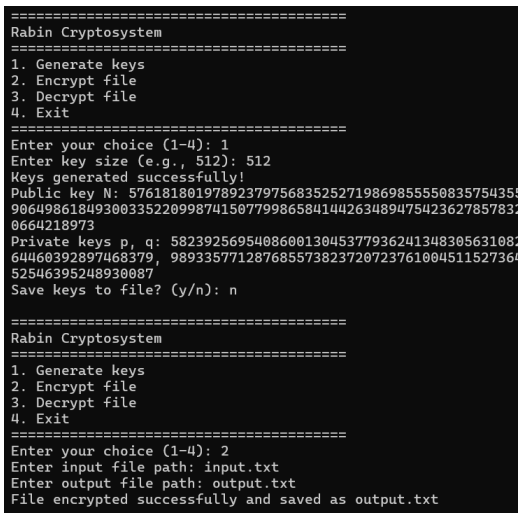

Рисунок 3.1 – Консольный интерфейс программного средства

Исходный текст, подлежащий зашифровке, приведен на рисунке 3.2.

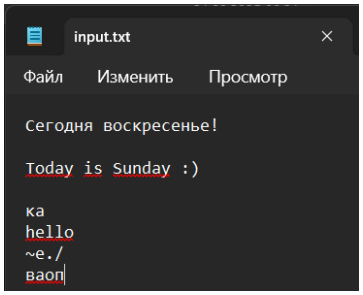

Рисунок 3.2 – Содержимое файла input.txt

Итоги криптографических преобразований приведены на рисунке 3.3.

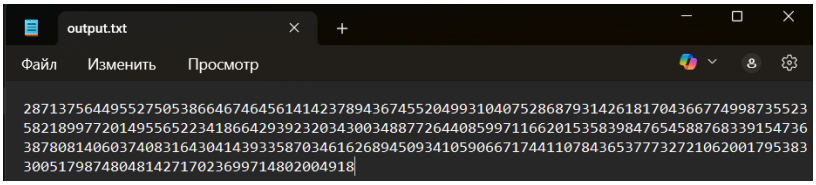

Рисунок 3.3 – Содержимое файла output.txt

Расшифровка ранее зашифрованного текста приведена на рисунке 3.4.

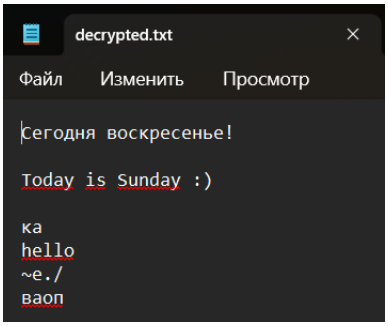

Рисунок 3.4 – Содержимое файла decrypted.txt

Тестирование программы проводилось на текстовых файлах различного объема и содержания. Во всех случаях обеспечивалось полное соответствие исходного и дешифрованного текста, что подтверждает корректность реализации алгоритма.

Разработанное программное средство демонстрирует практическую применимость криптосистемы Рабина для защиты конфиденциальной информации. Проведенные испытания подтверждают соответствие реализации теоретическим принципам асимметричного шифрования и возможность использования для решения задач защищенной передачи данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было реализовано программное средство, реализующее асимметричную криптосистему Рабина. Разработанное решение демонстрирует работоспособность данной криптографической системы и ее практическую применимость для защиты конфиденциальной информации. Реализация поддерживает полный цикл операций: генерацию ключевой пары, шифрование и дешифрование данных, а также работу с текстовыми файлами.

Проведенные испытания подтвердили корректность работы алгоритма – исходные текстовые данные полностью восстанавливаются после проведения полного цикла шифрования-дешифрования. Особое внимание было уделено решению проблемы неоднозначности дешифрования, характерной для криптосистемы Рабина, путем автоматического выбора корректного сообщения из четырех возможных вариантов. Реализованная система интерактивного консольного интерфейса обеспечивает удобство использования и может быть адаптирована для интеграции в процессы обработки защищаемой информации.

Полученный опыт реализации асимметричных криптографических алгоритмов представляет значительную ценность для понимания принципов современной защиты информации и может быть использован в дальнейшем для разработки более сложных систем информационной безопасности. Практическая реализация криптосистемы, основанной на сложности задачи факторизации больших чисел, подтверждает теоретические положения криптографии с открытым ключом и демонстрирует ее эффективность для обеспечения конфиденциальности данных.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
import random
from sympy import isprime


class RabinCryptosystem:
    def __init__(self):
        self.N = None
        self.p = None
        self.q = None

    def generate_keys(self, bits=512):
        while True:
            p = random.getrandbits(bits)
            if isprime(p) and p % 4 == 3:
                break

        while True:
            q = random.getrandbits(bits)
            if isprime(q) and q % 4 == 3:
                break

        self.N = p * q
        self.p = p
        self.q = q
        return self.N, self.p, self.q

    def encrypt(self, message):
        if self.N is None:
            raise ValueError("Keys not generated. Please generate keys
first.")

        m = int.from_bytes(message.encode('utf-8'), 'big')
        if m >= self.N:
            raise ValueError("The message is too large for the key size")

        ciphertext = pow(m, 2, self.N)
        return ciphertext

    def decrypt(self, ciphertext):
        if self.p is None or self.q is None:
            raise ValueError("Private keys not available. Please generate
keys first.")

        m_p = pow(ciphertext, (self.p + 1) // 4, self.p)
        m_q = pow(ciphertext, (self.q + 1) // 4, self.q)

        _, yp, yq = self.extended_gcd(self.p, self.q)
        N = self.p * self.q

        # Calculate all four possible roots
        r1 = (yp * self.p * m_q + yq * self.q * m_p) % N
        r2 = N - r1
        r3 = (yp * self.p * m_q - yq * self.q * m_p) % N
        r4 = N - r3

        # Try to decode each root
```

```
            for r in [r1, r2, r3, r4]:
                try:
                    decrypted_message = r.to_bytes((r.bit_length() + 7) // 8,
'big').decode('utf-8')
                    return decrypted_message
                except (UnicodeDecodeError, ValueError):
                    continue

            raise ValueError("Decryption failed: none of the roots produced
a valid message")

        # Extended Euclidean algorithm
        @staticmethod
        def extended_gcd(a, b):
            if a == 0:
                return b, 0, 1
            gcd, x1, y1 = RabinCryptosystem.extended_gcd(b % a, a)
            x = y1 - (b // a) * x1
            y = x1
            return gcd, x, y

        def save_keys(self, filename='key.txt'):
            if self.N is None or self.p is None or self.q is None:
                raise ValueError("Keys not generated")

            with open(filename, 'w') as f:
                f.write(f"{self.N}\n{self.p}\n{self.q}")

        def load_keys(self, filename='key.txt'):
            try:
                with open(filename, 'r') as f:
                    lines = f.readlines()
                    self.N = int(lines[0].strip())
                    self.p = int(lines[1].strip())
                    self.q = int(lines[2].strip())
            except (FileNotFoundError, IndexError, ValueError) as e:
                raise ValueError(f"Failed to load keys: {e}")


    class FileHandler:
        @staticmethod
        def read_file(file_path):
            with open(file_path, 'r', encoding='utf-8') as f:
                return f.read()

        @staticmethod
        def write_file(file_path, data):
            with open(file_path, 'w', encoding='utf-8') as f:
                f.write(str(data))


    class RabinApp:
        def __init__(self):
            self.crypto = RabinCryptosystem()
            self.file_handler = FileHandler()

        def display_menu(self):
            print("\n" + "="*40)
            print("Rabin Cryptosystem")
            print("="*40)
            print("1. Generate keys")
            print("2. Encrypt file")
```

```
            print("3. Decrypt file")
            print("4. Exit")
            print("="*40)

            choice = input("Enter your choice (1-4): ")
            return choice

        def generate_keys(self):
            try:
                bits = int(input("Enter key size (e.g., 512): "))
                N, p, q = self.crypto.generate_keys(bits)
                print(f"Keys generated successfully!")
                print(f"Public key N: {N}")
                print(f"Private keys p, q: {p}, {q}")

                save_choice = input("Save keys to file? (y/n): ").lower()
                if save_choice == 'y':
                    filename = input("Enter filename (default: key.txt): ")
or 'key.txt'
                    self.crypto.save_keys(filename)
                    print(f"Keys saved to {filename}")

            except ValueError as e:
                print(f"Error: {e}")

        def encrypt_file(self):
            try:
                input_file = input("Enter input file path: ")
                output_file = input("Enter output file path: ")

                message = self.file_handler.read_file(input_file)
                ciphertext = self.crypto.encrypt(message)

                self.file_handler.write_file(output_file, ciphertext)
                print(f"File   encrypted   successfully   and   saved   as
{output_file}")

            except Exception as e:
                print(f"Encryption error: {e}")

        def decrypt_file(self):
            try:
                input_file = input("Enter input file path: ")
                output_file = input("Enter output file path: ")

                ciphertext = int(self.file_handler.read_file(input_file))
                message = self.crypto.decrypt(ciphertext)

                self.file_handler.write_file(output_file, message)
                print(f"File   decrypted   successfully   and   saved   as
{output_file}")

            except Exception as e:
                print(f"Decryption error: {e}")

        def run(self):
            print("Rabin Cryptosystem")

            while True:
                choice = self.display_menu()

                if choice == "1":
```

```
                self.generate_keys()
            elif choice == "2":
                self.encrypt_file()
            elif choice == "3":
                self.decrypt_file()
            elif choice == "4":
                print("Bye bye!")
                break
            else:
                print("Invalid choice. Please try again.")


def main():
    app = RabinApp()
    app.run()


if __name__ == "__main__":
    main()
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Блок-схема алгоритма, реализующего программное средство